ABSTRACTION ISN'T THE ENTIRE STORY

CS 045

Computer Organization and Architecture

Prof. Donald J. Patterson

Adapted from Bryant and O'Hallaron,
Computer Systems:
A Programmer's Perspective, Third Edition

ABSTRACTION ISN'T THE ENTIRE STORY

OVERVIEW

- Course Theme
- Five Realities
- General System Walkthrough

ABSTRACTION ISN'T THE ENTIRE STORY

OVERVIEW

- Course Theme
- Five Realities
- General System Walkthrough

COURSE THEME

ABSTRACTION IS GOOD BUT DON'T FORGET REALITY

- Most Computer Science courses emphasize abstraction
  - Abstract data types
  - Asymptotic analysis
- These abstractions have limits
  - Especially in the presence of bugs
  - We need to understand details of underlying implementations
- Useful outcomes from taking 45
  - Become more effective programmers
  - Able to find and eliminate bugs efficiently
  - Able to understand and tune for program performance

ABSTRACTION ISN'T THE ENTIRE STORY

OVERVIEW

- Course Theme
- Five Realities
- General System Walkthrough

ABSTRACTION ISN'T THE ENTIRE STORY

OVERVIEW

- Course Theme
- Five Realities
- General System Walkthrough

REALITY #1

INTS ARE NOT INTEGERS
FLOATS ARE NOT REALS

- Example 1: Is $x^2 \geq 0$?

```c
#include <stdio.h>

int main(){

    float y = 40000.0;
    printf("float\n");
    printf("%f squared is %f\n",y,y*y);
    y = 50000.0;
    printf("%f squared is %f\n",y,y*y);

    int x = 40000;
    printf("int\n");
    printf("%d squared is %d\n",x,x*x);
    x = 50000;
    printf("%d squared is %d\n",x,x*x);
}
```

REALITY #1

INTS ARE NOT INTEGERS
FLOATS ARE NOT REALS

- Example 1: Is $x^2 \geq 0$?

```
#include <stdio.h>

int main(){
    float
    40000.000000 squared is 1600000000.000000
    50000.000000 squared is 2500000000.000000
    y = 50000.0;
    printf("%f squared is %f\n",y,y*y);

    int x = 40000;
    printf("int\n");
    printf("%d squared is %d\n",x,x*x);
    x = 50000;
    printf("%d squared is %d\n",x,x*x);
}
```

REALITY #1

INTS ARE NOT INTEGERS
FLOATS ARE NOT REALS

- Example 1: Is $x^2 \geq 0$?

```
#include <stdio.h>

int main(){
  float
  40000.000000 squared is 1600000000.000000
  50000.000000 squared is 2500000000.000000
    y = 50000.0;
  int
  40000 squared is 1600000000
  50000 squared is -1794967296
    printf("%d squared is %d\n",x,x*x);
    x = 50000;
    printf("%d squared is %d\n",x,x*x);
}
```

REALITY #1

INTS ARE NOT INTEGERS
FLOATS ARE NOT REALS

- Example 1: Is $x^2 \geq 0$?

  - Floats?

```
float
40000.000000 squared is 1600000000.000000
50000.000000 squared is 2500000000.000000
```

- Ints?

```
int
40000 squared is 1600000000
50000 squared is -1794967296
```

REALITY #1

INTS ARE NOT INTEGERS
FLOATS ARE NOT REALS

- Example 1: Is $x^2 \geq 0$?
  - Floats?
    - Yes!
  - Ints?

```
int
40000 squared is 1600000000
50000 squared is -1794967296
```

REALITY #1

INTS ARE NOT INTEGERS
FLOATS ARE NOT REALS

- Example 1: Is $x^2 \geq 0$?
  - Floats?
    - Yes!
  - Ints?
    - 40,000 * 40,000 -> 1,600,000,000
    - 50,000 * 50,000 -> No (why?)

REALITY #1

INTS ARE NOT INTEGERS
FLOATS ARE NOT REALS

- Example 2: $(x + y) + z \stackrel{?}{=} x + (y + z)$

```c
#include <stdio.h>

int main(){
    int x = -10;
    int y = 10;
    int z = 30;
    printf("int\n");
    printf("(%d + %d) + %d = %d\n",x,y,z,(x+y)+z);
    printf("%d + (%d + %d) = %d\n",x,y,z,x+(y+z));

    float fx = 1e20;
    float fy = -1e20;
    float fz = 3.1415;
    printf("float\n");
    printf("(%e + %e) + %f = %f\n",fx,fy,fz,(fx+fy)+fz);
    printf("%e + (%e + %f) = %f\n",fx,fy,fz,fx+(fy+fz));
}
```

REALITY #1

INTS ARE NOT INTEGERS
FLOATS ARE NOT REALS

- Example 2: $(x + y) + z \stackrel{?}{=} x + (y + z)$

```
#include <stdio.h>

int main(){
    int
    (-10 + 10) + 30 = 30
    -10 + (10 + 30) = 30
    printf("(%d + %d) + %d = %d\n",x,y,z,(x+y)+z);
    printf("%d + (%d + %d) = %d\n",x,y,z,x+(y+z));

    float fx = 1e20;
    float fy = -1e20;
    float fz = 3.1415;
    printf("float\n");
    printf("(%e + %e) + %f = %f\n",fx,fy,fz,(fx+fy)+fz);
    printf("%e + (%e + %f) = %f\n",fx,fy,fz,fx+(fy+fz));
}
```

REALITY #1

INTS ARE NOT INTEGERS
FLOATS ARE NOT REALS

- Example 2: $(x + y) + z \stackrel{?}{=} x + (y + z)$

```
#include <stdio.h>

int main(){
    int x = 10;
    int y = -10;
    int z = 30;
    printf("int\n");
    printf("(%d + %d) + %d = %d\n",x,y,z,(x+y)+z);
    printf("%d + (%d + %d) = %d\n",x,y,z,x+(y+z));
    float fx = 1.000000e+20;
    float fy = -1.000000e+20;
    float fz = 3.1415;
    printf("float\n");
    printf("(%e + %e) + %f = %f\n",fx,fy,fz,(fx+fy)+fz);
    printf("%e + (%e + %f) = %f\n",fx,fy,fz,fx+(fy+fz));
}
```

```
int
(-10 + 10) + 30 = 30
-10 + (10 + 30) = 30
```

```
float
(1.000000e+20 + -1.000000e+20) + 3.141500 = 3.141500
1.000000e+20 + (-1.000000e+20 + 3.141500) = 0.000000
```

REALITY #1

INTS ARE NOT INTEGERS
FLOATS ARE NOT REALS

- Example 2: $(x + y) + z \stackrel{?}{=} x + (y + z)$

  - Ints?

```
int
(-10 + 10) + 30 = 30
-10 + (10 + 30) = 30
```

- Floats?

```
float
(1.000000e+20 + -1.000000e+20) + 3.141500 = 3.141500
1.000000e+20 + (-1.000000e+20 + 3.141500) = 0.000000
```

REALITY #1

INTS ARE NOT INTEGERS
FLOATS ARE NOT REALS

- Example 2: $(x + y) + z \stackrel{?}{=} x + (y + z)$

  - Ints?
    - Yes!

  - Floats?

```
float
(1.000000e+20 + -1.000000e+20) + 3.141500 = 3.141500
1.000000e+20 + (-1.000000e+20 + 3.141500) = 0.000000
```

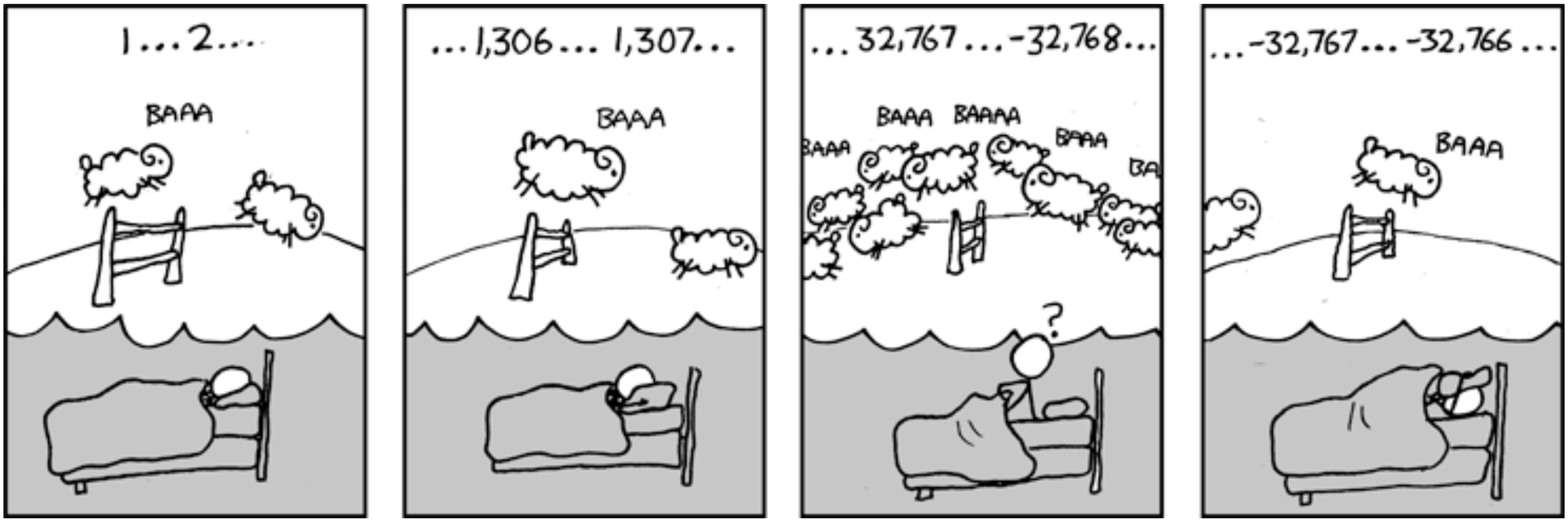
REALITY #1

INTS ARE NOT INTEGERS
FLOATS ARE NOT REALS

- Example 2: $(x + y) + z \stackrel{?}{=} x + (y + z)$
  - Ints?
    - Yes!
  - Floats?
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> No (why?)

REALITY #1

INTS ARE NOT INTEGERS
FLOATS ARE NOT REALS



- We need to understand what is happening here

Source: xkcd.com/571

REALITY #1

WE CANNOT ASSUME TYPICAL MATH PROPERTIES

- Because of finiteness of representations
- Integer operations satisfy “ring” properties
  - Commutativity, associativity, distributivity
- Floating point operations satisfy “ordering” properties
  - Monotonicity, values of signs
- Observation
  - Need to understand which abstractions apply in which contexts
  - Important issues for compiler writers and serious application programmer

REALITY #2

YOU NEED TO KNOW ABOUT ASSEMBLY

- Chances are, you'll never write programs in assembly
  - Compilers are much better & more patient than you are
- But: Understanding assembly is key to machine-level execution model
  - Behavior of programs in presence of bugs
    - High-level language models break down
  - Tuning program performance
    - Understand optimizations done / not done by the compiler
    - Understanding sources of program inefficiency

REALITY #2

YOU NEED TO KNOW ABOUT ASSEMBLY

- But: Understanding assembly is key to machine-level execution model
  - Implementing system software
    - Compiler has machine code as target
    - Operating systems must manage process state
  - Creating / fighting malware
    - x86 assembly is our language of choice
    - arm is a close good second

REALITY #3

MEMORY MATTERS

- Random Access Memory Is an Unphysical Abstraction
- Memory is not unbounded
  - It must be allocated and managed
  - Many applications are memory dominated
- Memory referencing bugs especially pernicious
  - Effects are distant in both time and space
- Memory performance is not uniform
  - Cache and virtual memory effects can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements

REALITY #3

MEMORY REFERENCING BUG EXAMPLE

```c
#include <stdio.h>

typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}


int main(){
    printf("fun(%d) -> %lf\n",0,fun(0));
    printf("fun(%d) -> %lf\n",1,fun(1));
    printf("fun(%d) -> %lf\n",2,fun(2));
    printf("fun(%d) -> %lf\n",3,fun(3));
    printf("fun(%d) -> %lf\n",4,fun(4));
    printf("fun(%d) -> %lf\n",5,fun(5));
    printf("fun(%d) -> %lf\n",6,fun(6));
}
```

REALITY #3

MEMORY REFERENCING BUG EXAMPLE

```c
#include <stdio.h>

typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}


int main(){
    printf("fun(%d) -> %lf\n",0,fun(0));
    printf("fun(%d) -> %lf\n",1,fun(1));
    printf("fun(%d) -> %lf\n",2,fun(2));
    printf("fun(%d) -> %lf\n",3,fun(3));
    printf("fun(%d) -> %lf\n",4,fun(4));
    printf("fun(%d) -> %lf\n",5,fun(5));
    printf("fun(%d) -> %lf\n",6,fun(6));
}
```

```
fun(0) -> 3.140000
fun(1) -> 3.140000
fun(2) -> 3.140000
fun(3) -> 2.000001
fun(4) -> 3.140000
fun(5) -> 3.140000
Segmentation fault
```

REALITY #3

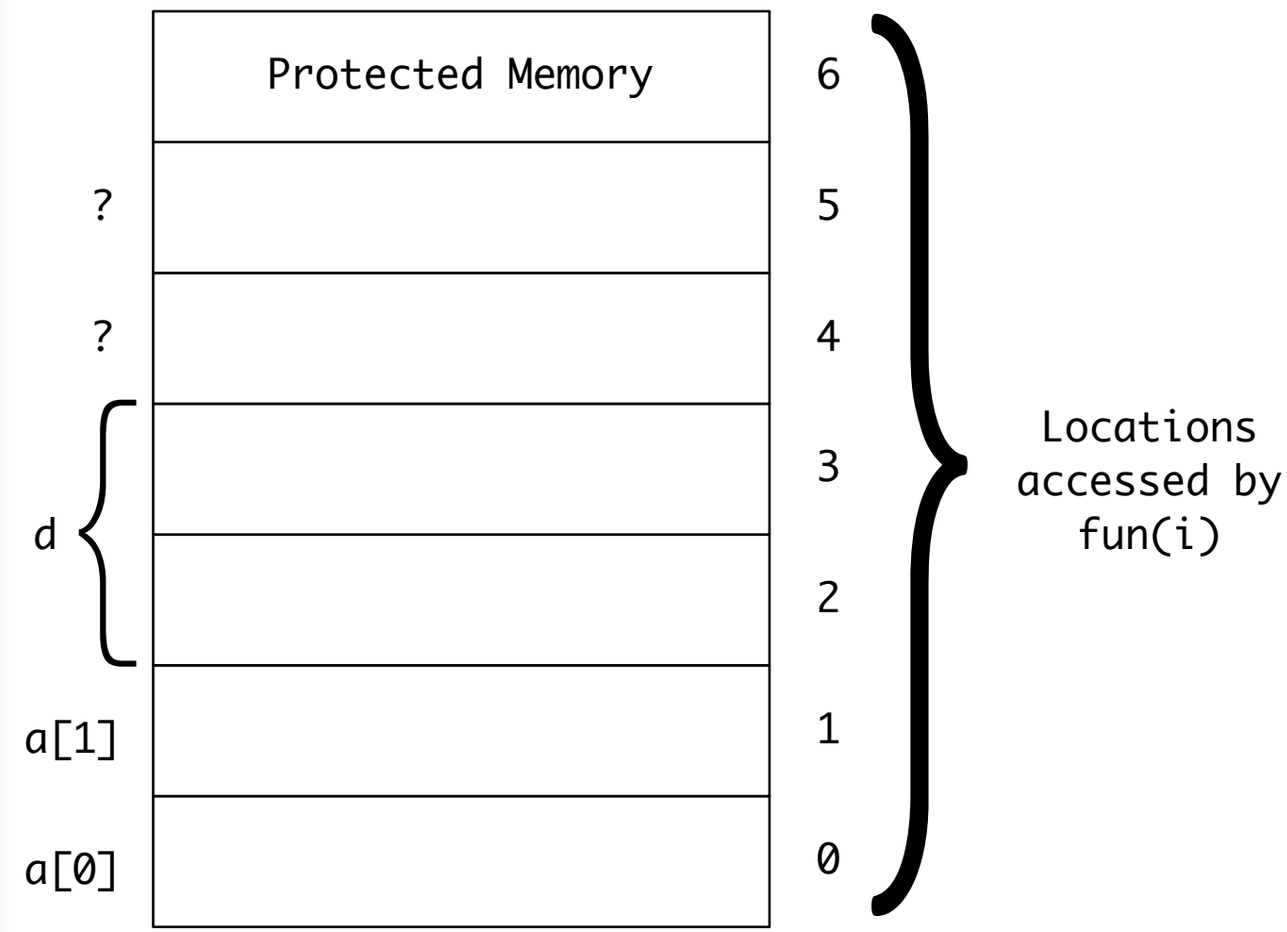
MEMORY REFERENCING BUG EXPLANATION

```c
#include <stdio.h>

typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}


int main(){
    printf("fun(%d) -> %lf\n",0,fun(0));
    printf("fun(%d) -> %lf\n",1,fun(1));
    printf("fun(%d) -> %lf\n",2,fun(2));
    printf("fun(%d) -> %lf\n",3,fun(3));
    printf("fun(%d) -> %lf\n",4,fun(4));
    printf("fun(%d) -> %lf\n",5,fun(5));
    printf("fun(%d) -> %lf\n",6,fun(6));
}
```



REALITY #3

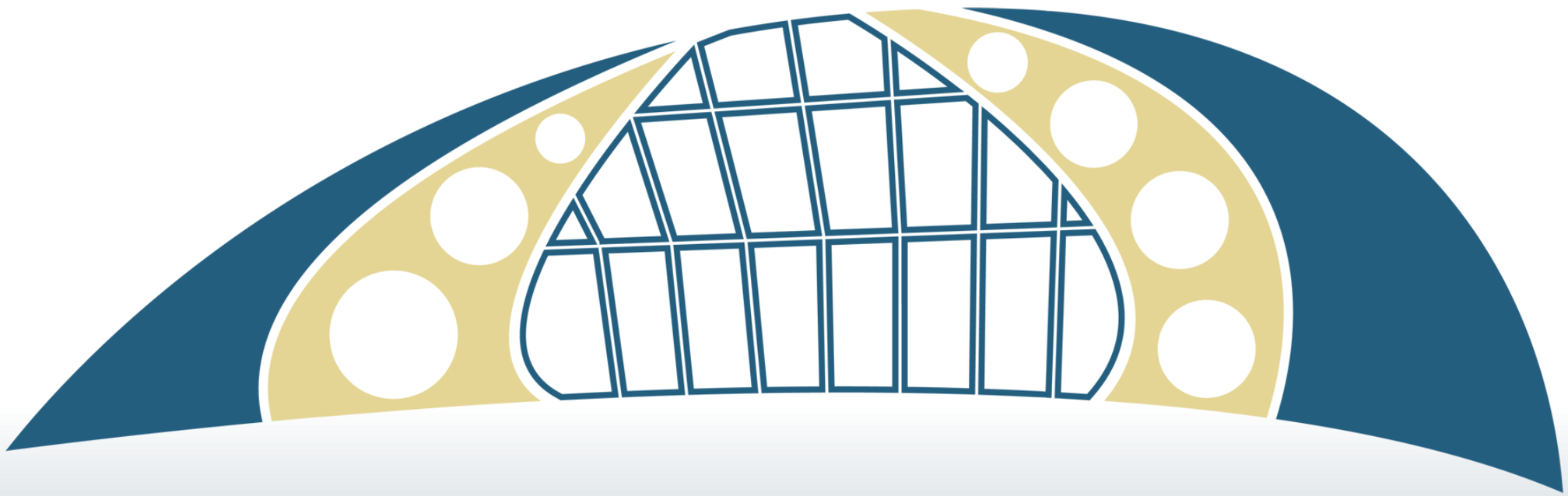
MEMORY MATTERS

- C and C++ do not provide any memory protection
  - Out of bounds array references
  - Invalid pointer values
  - Abuses of malloc/free
- Can lead to nasty bugs
  - Whether or not bug has any effect depends on system and compiler
  - Action at a distance
    - Corrupted object logically unrelated to one being accessed
    - Effect of bug may be first observed long after it is generated

REALITY #3

MEMORY MATTERS

- How can I deal with this?
  - Program in Java, Ruby, Python, Erlang, …
  - Understand what possible interactions may occur
  - Use or develop tools to detect referencing errors (e.g. Valgrind)



WESTMONT INSPIRED

COMPUTING LAB